Minimizing Oscillatory Signals in Deep Reinforcement Learning Control of Unmanned Undersea Vehicles

Christopher Hixenbaugh

Alfa Heryudono

Eugene Chabot

TABLE OF CONTENTS

Section **Page**

LIST OF ILLUSTRATIONS

Figure **Page**

LIST OF TABLES

Table **Page**

[This page will continue the lists or be numbered as a reverse blank.]

1. ABSTRACT

Control systems for Unmanned Undersea Vehicles (UUVs) are typically implemented using Proportional Integral Derivative (PID) control systems. PID control systems for UUVs are resource-intensive to tune since they require engineers, marine operators, and ship crew working together to adjust the controller. Furthermore, PID controllers rely heavily on complex dynamical system models that contain assumptions to reduce the computational complexity of the models. The controller's performance may degrade if environmental and external conditions do not fully align with those assumptions. In this work, a Deep Reinforcement Learning (DRL) control system based on the Deep Deterministic Policy Gradient (DDPG) algorithm is studied for a UUV control system. The DDPG algorithm is model-free, meaning that the explicit formulations of the complex dynamical system models are optional to provide optimal performance. The DRL-based control systems are tuned autonomously, reducing the resources needed for manual tuning. Our focus is to study how different Deep Neural Network (DNN) architectures implemented as part of the DDPG agent affect the control signal output by the control system. DNN architectures that minimize undesirable oscillations in the control signals, which could potentially cause physical damage to the UUV, will be of interest. Numerical case studies will be presented.

2. INTRODUCTION

Technologies that rely entirely on autonomous systems have played significant roles in advancing environmental research in the undersea domain. Unmanned Undersea Vehicles (UUVs) such as the Naval Post Graduate School UUV research platform have played a role in advancing the state of the art of autonomous systems for research purposes. Using autonomous systems for research is becoming more popular because autonomous systems can relieve humans from repetitive tasks and reduce the risk of injury. Additionally, UUVs can be manufactured in large quantities at relatively lower costs. Moreover, due to advances in computing and battery technologies, UUVs can undertake more extended missions without human interventions.

One of the essential parts of UUVs is the control system. UUV control system configuration may change based on the vehicle payload or environmental factors such as salinity. The control system is responsible for achieving and maintaining stable flight about at a target path. PID controllers are widely implemented on UUVs, although their use comes with a significant cost to tune the controller. The steep cost does not provide the benefits of a robust or intelligent solution because of two major problems.

The first problem is that PID controllers rely on complex dynamic system models to control the UUV. The dynamic system models have simplifying assumptions that allow the control problem to be solved efficiently. When the assumptions are not valid, a PID controller can provide sub-optimal control, or even complete loss of control can occur. The second problem is that PID controllers are not intelligent and cannot learn autonomously. PID controllers require multiple engineers and other personnel to spend days collecting and analyzing data to tune the controller. Tuning a PID controller is a manual task that introduces the opportunity for human error.

There is much ongoing research in using Deep Reinforcement Learning methods for autonomous vehicle control systems, and it has shown promising results [1, 2]. Deep Reinforcement

Learning controllers have been shown to outperform PID controllers for UUVs executing path-following missions [3]. Additionally, Deep Reinforcement Learning based controllers have been demonstrated to provide superior attitude control compared to PID controllers for Unmanned Aerial Vehicles (UAVs) [4-5]. Although this example is not specific to UUVs, this concept from the aerial domain can be translated to the undersea domain.

Some of the most popular Deep Reinforcement Learning algorithms being used for autonomous vehicle control system development are the Proximal Policy Optimization (PPO) [6] and Deep Deterministic Policy Gradient (DDPG) [7] algorithms. This study will focus on the DDPG algorithm. The DDPG algorithm is an Actor-Critic type Deep Reinforcement Learning algorithm. Actor-Critic algorithms learn both a policy and value function. The concept of an Actor-Critic algorithm is that the policy function (the actor) determines the actions of the system according to the current state, and the value function (the critic) critiques the actions. The DDPG algorithm uses the state-action value function. In Deep Reinforcement learning, the policy and value functions are approximated by DNNs, specifically Multi-Layer Perceptrons (MLPs) in this study.

There are two major benefits that a Deep Reinforcement Learning controller based on the DDPG algorithm provides compared to a traditional PID controller for UUVs. The first benefit is that the DDPG algorithm is model-free. It does not require any knowledge of the vehicle or environmental dynamics to provide optimal control. Therefore, it avoids the downfalls of simplifying assumptions needed to solve complex vehicle or environmental dynamic system models efficiently. Secondly, Deep Reinforcement Learning based control systems can be tuned (trained) autonomously. This will reduce the resources needed to tune a Deep Reinforcement Learning based control system compared to a PID control system.

An area of ongoing research of Deep Reinforcement Learning based control systems is training the Reinforcement Learning agent to provide optimal control with a smooth control signal that is free of high frequency and high amplitude oscillations. Much of the research in this area is focused on reward signal engineering and algorithmic improvements [5,8]. The contribution of this work will be examining the effects of different DNN architectures on the control signal behavior produced by a DDPG agent. The effects of DNN architectures on Deep Reinforcement Learning agent performance for the continuous control problem of real-world systems are often overlooked in the literature. This study will be an attempt to fill that void.

In our study, a three degree of freedom simulation environment was created in Simulink where the Naval Post Graduate School Autonomous Undersea Vehicle (NPSUUV) model [9] from the Marine Systems Simulator (MSS) Matlab Toolbox [10] is implemented as the UUV model to execute a path following the task of a curved surface. The NPSUUV model is tasked with keeping a constant altitude above a curved surface while maintaining a pitch angle that is as close to parallel to the surface as possible at a constant speed. A DDPG agent is used to control the UUV's control surfaces to provide altitude and pitch angle control to accomplish this task. While keeping the DDPG agent hyperparameters unchanged, this study will investigate the effects of DNN architectures commonly found in Deep Reinforcement Learning literature on the control signal behavior. The DNN architectures that will be explored are MLPs with the following layer and node counts for both actor and critic networks: [64, 64], [400, 300], [256, 256], and [100, 50, 25].

The remainder of the paper is structured as followed. The Problem Formulation section will provide a brief background on the NPSUUV dynamics, PID control, the DDPG algorithm, DNN categories used for Deep Reinforcement Learning, and a brief background on methods to reduce the oscillatory characteristics of control signals produced by Reinforcement Learning agents. The Experimental Analysis section will describe the setup and results of the numerical experiments run in this study. The overall work and future planned work will be described in the Conclusions and Future Work section.

3. PROBLEM FORMULATION

3.1 NPSUUV DYNAMICS

The UUV numerical model used in this study is the Naval Post Graduate School Aries UUV model as described in [9]. The authors detail a six degree of freedom dynamic system model for this UUV. The equations of motion for this system are developed in the body fixed reference frame as described in [11]. The velocity components of the UUV in the body fixed reference frame are described as:

$$\dot{x} = [u(t), v(t), w(t), p(t), q(t), r(t)]$$

Where *u(t)* is the surge velocity, *v(t)* is the sway velocity, *w(t)* is the heave velocity, *p(t)* is the roll velocity, *q(t)* is the pitch velocity, and *r(t)* is the yaw velocity.

The body-reference frame for the NPSUUV is illustrated in **Figure 1.**

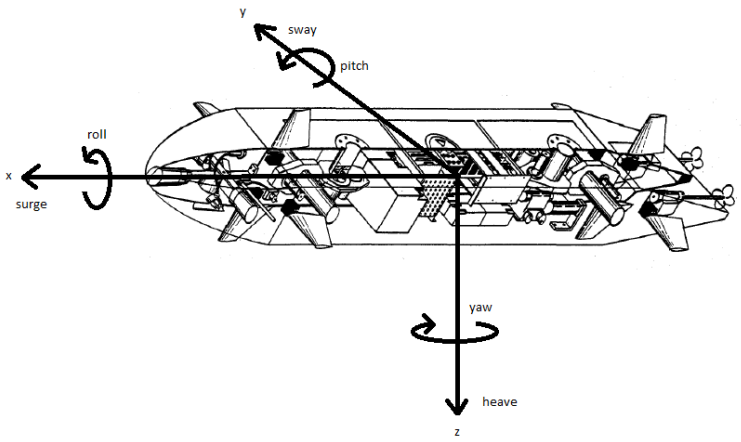

Figure 1- Body-reference frame for Aries UUV

The six components describing UUV position are:

$$x = [x(t), y(t), z(t), \phi(t), \theta(t), \psi(t)]$$

Where *x(t)* is the position in the surge direction, *y(t)* is the position in the sway direction, *z(t)* is the position in the heave direction, $\phi(t)$ is the roll angle, $\theta(t)$ is the pitch angle, and $\psi(t)$ is the yaw angle.

To control the UUV, the model control inputs are:

$$ui = [\delta_r(t), \delta_s(t), \delta_b(t), \delta_{bp}(t), \delta_{bs}(t), n]$$

Where $\delta_r(t)$ is the rudder angle, $\delta_s(t)$ is the port and starboard stern plane angle, $\delta_b(t)$ is the top and bottom bow plane angle, $\delta_{bp}(t)$ is the port bow plane angle, $\delta_{bs}(t)$ is the starboard bow plane, and *n* is the propeller shaft speed.

The equation of motion for the UUV is described in terms of twelve non-linear systems of equations as described in [12]:

$$M(t)\frac{dx}{dt} = \boldsymbol{f}\big(x(t), z(t), c(t)\big) + \boldsymbol{g}\big(x(t), z(t)\big) * \boldsymbol{u}(t)$$

$$\frac{dz}{dt} = \boldsymbol{h}(z(t), x(t), u_c)$$

The mass matrix *M(t)* includes both the mechanical and hydrodynamic added mass. The functions **f** and **g** are mappings of the UUV motions into forces such as Coriolis, gravitational, centrifugal, hydrostatic, hydrodynamic, and moments acting on the UUV with coefficients *c*, and the motion dependent influence of the control surfaces, propeller, and ballasting. The function **h** includes the kinematical relationships found in performing the coordinate system transformation between the body reference frame, inertial reference frame, and ocean current u_c.

The equations of motion for each degree of freedom can be found in the appendix section and are implemented similarly in Matlab MSS toolbox *npsauv.m* function

3.2 PROPORTIONAL INTEGRAL DERIVATIVE CONTROLLER

Proportional Integral Derivative (PID) controllers are simple and versatile controllers that are widely used in practice. Although PID controllers are easy to implement and tune, they are very sophisticated in that these controller types capture the history of the system through integration and can anticipate the future behavior of the system through differentiation [13].

The basic architecture of a feedback control loop with a PID controller is illustrated in **Figure 2.**

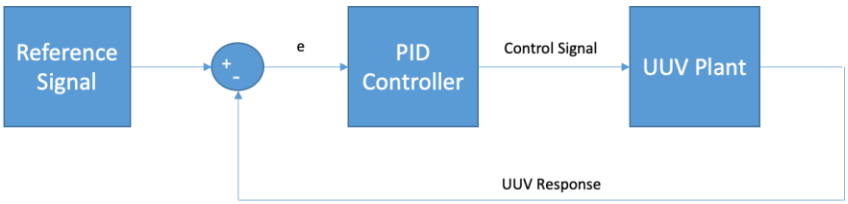

Figure 2- Feedback control loop with PID controller

In this model, the reference signal is the target response that we desire the UUV to achieve. The PID controller accepts the error (e) between the reference signal and the UUV response and computes a control signal for the UUV that will minimize the difference between these two signals.

The control signal is computed by the PID controller by considering the error, the integral of the error, and the derivative of the error as follows:

$$Control\ Signal\ =\ C_p e(t) + C_i \int e(t)dt + C_d \frac{de(t)}{dx}$$

The coefficient C_p scales the control signal proportionally to the error and influences how quickly the system reacts to minimizing the error. C_i scales the integral of the error and helps minimize the steady state error. C_d scales the derivative of the error and allows the controller to predict the future error. Coefficients C_p, C_i, and C_d are determined empirically through experimentation.

The empirical nature of determining (tuning) PID controller coefficients is both advantageous and detrimental. While the controller easy to understand, it is very resource intensive to tune. Tuning a UUV PID controller requires several engineers to spend time offshore with a physical UUV to manually tune the coefficients through experimentation. After the resource intensive tuning effort, a PID controller is model based, meaning that a dynamic system model of the UUV is required to provide control. This type of dynamic system model is very complex and requires simplifying assumptions to be solvable in a computationally acceptable manner. If the UUV encounters an environmental condition that conflicts with a simplifying assumption, it can cause the controller to provide sub-optimal control, or even complete loss of control.

To improve control system technology for UUVs, there is ongoing research on using Deep Reinforcement Learning methods for UUV control. Key benefits of using Deep Reinforcement Learning methods for UUV control have been described in the introduction section of this paper.

3.3 DDPG ALGORITHM

This study will focus on using the DDPG algorithm for control of the UUV described in this section. The DDPG algorithm is an off-policy, model-free actor-critic Deep Reinforcement Learning algorithm. Model-free means that the algorithm does not rely on an environmental or UUV dynamic system model to achieve optimal performance. The DDPG algorithm is an off-policy algorithm. This means that the state-action function does not depend on the policy used to gather experiences. An off-policy algorithm considers the maximum Q value over all the

potential actions available in a given a state [14]. This is different from an on-policy algorithm such as SARSA [15] which relies on the Q value calculated by the experience gathering policy.

A benefit of off-policy algorithms is that a large number of past experiences can be considered when computing the Q value. These experiences given by the tuple, (s_t, a_t, r_t, s_{t+1}) , are stored in an experience buffer, or replay buffer, in a First In First Out (FIFO) fashion. s_t is the state at time t, a_t is the action selected by the agent at time t, r_t is the reward at time t, and s_{t+1} is the next state. As the buffer becomes full, the oldest experiences are removed. Removing old experiences is important because older experiences tend to be less useful and it's preferable for the agent to learn from more recent experiences. Each time the agent trains, experiences are sampled from a uniform distribution to update the parameters of the critic network. The number of experiences used during training is called the batch size. The buffer size and batch size are tunable DDPG agent hyperparameters.

The DDPG algorithm is an actor-critic type algorithm. In Deep Reinforcement Learning the actor and critic are implemented as DNNs. The actor network, $\mu(s|\theta)$, learns a parameterized policy that computes an action according to the current state. The critic network, $Q(s, a|\phi)$, learns a value function given a state action pair and provides reinforcing information to the actor. θ and ϕ are the weights of the actor and critic networks respectively. The critic computes the temporal difference (TD) error that is used in both the actor and critic networks during the training process. A high level architecture of an actor-critic type Deep Reinforcement Learning agent is found in **Figure 3.**

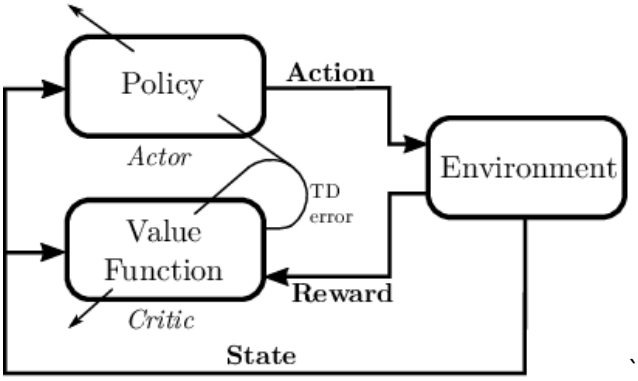

Figure 3- Actor-critic agent architecture [16]

There are two actor and critic networks in the DDPG algorithm. There is a trained network and a target network for both actor and critic. The target networks for the actor and critic are denoted by $\mu'(s|\theta')$ and $Q'(s, a|\phi')$ where θ' are the weights of the target actor network and ϕ' are the weights of the target critic network. The reason for using two networks is that it stabilizes training [17]. During training, the actor and critic networks are being updated frequently and this makes training difficult because there can be large changes to the actor and critic networks between each training step. To mitigate this, target networks are implemented that are updated at a slower rate. There are two strategies to update the weights of the target network. One strategy is to copy the weights from the actor and critic networks to their respective target networks periodically, or to use the Polyak averaging method to update the target actor and critic weights at a slower constant rate. The Polyak method will be used in this study because it eliminates the potential for large changes in the target actor and critic networks that may occur if the time

between updates is too long. The Polyak update method for the target actor and critic networks are:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta'$$

$$\phi' \leftarrow \tau\phi + (1-\tau)\phi'$$

τ is a tunable hyperparameter called the Target Smoothing Factor that controls how fast the target actor and critic network weights change with respect to the actor and critic networks. Typical values for τ are on the order of 10^{-3}. The target actor and critic networks are updated during each time step during training just like the actor and critic networks.

The actor network is updated by sampling the policy gradient:

$$\nabla_\theta \mathrm{J} \approx \left(\frac{1}{N}\sum_i \nabla_a Q(s_i, \mu(s_i|\phi)\nabla_\theta \mu(s_i|\theta)\right)$$

The critic network is updated by minimizing the loss:

$$L = \frac{1}{N}\sum_i \big(y_i - Q(s_i, a_i|\phi)\big)^2$$

To promote exploration in the DDPG algorithm, an exploration policy is built by adding noise to the action selected by the actor network. The noise added to the policy is defined by the Ornstein-Uhlenbeck process [18]. The exploration policy is defined as

$$\mu'(s) = \mu(s|\theta) + OUnoise$$

The Ornstein-Uhlenbeck (OU) noise is implemented as [19]:

$$x_t = x_{t-1} + \vartheta(\mu - x_{t-1})dt + randn\big(size(\mu)\big)\sigma\sqrt{dt}$$

Where x_t is the noise to be added to the selected action at time t, ϑ is the Mean Attraction Constant which specifies how quickly the noise model output is attracted to the noise model mean, μ. σ is the standard deviation and is defined as a percentage of the action space range.

$$\sigma = 0.X(action\_space_{max} - action\_space_{min})$$

During each time step, the standard deviation is decayed by the decay rate ε. The decayed standard deviation is

$$\sigma_{decayed} = \sigma(1 - \varepsilon)$$

The standard deviation to be used in the next step is the calculated as

$$\sigma = \max\big(\sigma_{decayed}, \sigma_{min}\big)$$

where σ_{min} is the minimum standard deviation for the noise model. σ_{min} is a tunable hyperparameter.

3.4 DEEP NEURAL NETWORK CATEGORIES FOR DEEP REINFORCEMENT LEARNING

DNNs are used to approximate the policy and state-action value functions for the DDPG algorithm. The three main categories of DNN's are Multi-Layer Perceptrons (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). MLP's are general purpose DNNs that generally perform well for low dimensional problems where data sequence or spatial considerations are not important. CNN's perform very well at learning from images because they are designed to take advantage of the spatial nature of image data. The strength of RNNs is learning from sequences of information such as text or speech data.

Hybrid networks can also be constructed which are built with some combination of MLPs, CNNs, or RNNs and are suitable for when model inputs contain a mixture of data types. For example, if a network's input was UUV state information and image data, a hybrid network consisting of a MLP and CNN may be advantageous to process these different data types.

The type of DNN that this study implements is the MLP. MLP's are a suitable choice for this problem because the policy and state-action value functions being approximated are a function of the states of the UUV and the actions selected by the DDPG agent. There is no spatial structure or information sequence to be considered. If there was an image input for the actor or critic networks, a CNN would be a better choice of network as a strength of CNNs is learning from images.

3.5 OSCILLATORY CONTROL SIGNALS

Deep Reinforcement Learning agents have been shown to be able to learn and perform optimal control for UAVs and UUVs. There are two ways to achieve optimal control. The preferred way to achieve optimal control for UUVs is by a completely smooth control signal that minimizes the error between the UUV response and reference signal. Optimal control can also be achieved through bang-bang control. Bang-bang control is where optimal control is achieved by high frequency, high amplitude oscillations between the upper and lower bounds of the control signal. These two optimal control signal descriptions represent the extremes. Deep Reinforcement Learning agents can also learn to provide optimal control with control signals that are relatively smooth, but still exhibit some oscillatory characteristics.

For this type of problem, the goal is to always achieve optimal control with a completely smooth control signal. Removing oscillations from the control signal will prevent damage to a system that high frequency, high amplitude oscillations in the control signal can cause as well as preserve sensor data quality by minimizing unwanted changes to the UUVs attitude.

Methods to achieve a completely smooth control signal for Deep Reinforcement Learning agents is an active area of research. A widely used approach to solving this problem is through reward

signal engineering where the agent is penalized for oscillations of the control signal such as in [5]. Another approach in a recent study has shown promising results by regularizing the policy function to smooth the control signal [8].

While both approaches have been proven effective in practice or benchmark testing, there is another aspect to this problem that is overlooked in the literature and is an area that this study aim to address. Little attention has been paid to how DNN architecture influences the control signal independent of the Deep Reinforcement Learning agent configuration for a real world problem. This study implements a reward signal engineering approach to reducing the oscillations in the control signal and will examine the effects of DNN architectures on the control signal while keeping all parameters of the Reinforcement Learning agent constant.

4. EXPERIMENTAL ANALYSIS

This section will describe the configuration of the control system architecture, numerical simulation environment and DDPG agent configuration used to execute numerical experiments of this study. We will also present the results of our experiments in this section.

4.1 EXPERIMENTAL SETUP

The goal of the control system is for the UUV to maintain a fixed altitude of 10m above the curved surface defined by the equation:

$$S = 2\sin\left(0.0209x + \frac{\pi}{2}\right) - 40$$

to within 0.1m while also maintaining the UUV's pitch angle to be within 4° of the angle of the curved surface at any given time. This will keep the UUV approximately parallel to the surface. The UUV's speed is constant.

The Deep Reinforcement Learning control system for the NPSUUV was developed using Matlab/Simulink 2020a. The feedback control system architecture is illustrated in **Figure 4.**

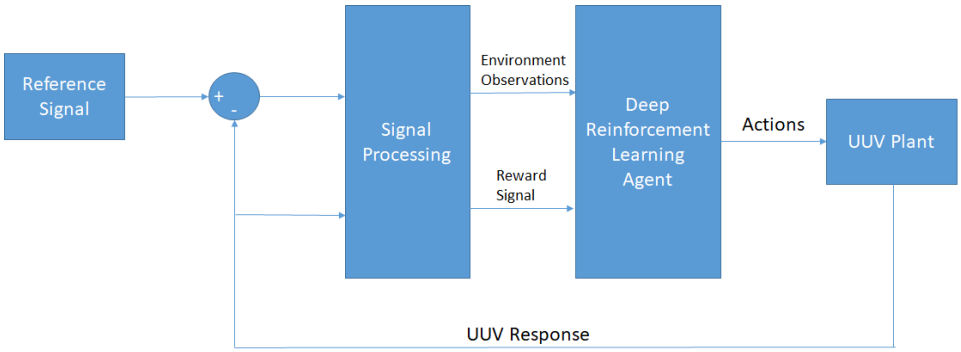

Figure 4- Feedback control loop for DDPG Reinforcement Learning control system for the NPSUUV.

The reference signal block defines the target altitude and pitch angle that the UUV is to achieve. The UUV Plant block is the dynamic system model to compute the UUV's response to the actions selected by the Deep Reinforcement Learning Agent block. The UUV Plant is the NPSUUV model described in the previous section. The Deep Reinforcement Learning agent is the DDPG algorithm as implemented in Matlab/Simulink 2020a Reinforcement Learning toolbox [20]. The Signal Processing block computes the environment observations (states) and reward signal.

The observations generated by the signal processing block are:

$$s_t = \{\Delta z_t, \sin\Delta\theta_t, \cos\Delta\theta_t, \frac{d\Delta\theta_t}{dt}, sin\theta_{ref,t}, cos\theta_{ref,t}, w_t, q_t, sin\theta_t, cos\theta_t\}$$

Where $\Delta z = z_{ref} - z$ and $\Delta\theta = \theta_{ref} - \theta$. The angle is divided into sine and cosine components to avoid issues with periodicity. The reference altitude, z_{ref}, is defined as altitude above the surface rather than depth from the sea surface. $z_{ref} = z_{surface} + z_{altitude}$

The reward signal generated by the signal processing block is:

$$r_t = -\left(c_1 {\Delta z_t}^2 + \mathrm{c}_2 {\Delta\theta_t}^2 + \mathrm{c}_3 p_t^2 + \mathrm{c}_4 (N_t - N_{t-1})^2\right) + A_t + P_t + B_t$$

Where N_t is the control signal, $A_t = 1$ when ${\Delta z_t}^2 \leq 0.01\ m^2$, $A_t = 0$ otherwise, $P_t = 1$ when ${\Delta\theta_t}^2 \leq 0.00524\ rad^2$, $P_t = 0$ otherwise, and $B_t = -10000$ when $z \leq z_{surface}$ or $z \geq 0$ which would signify that the UUV has either collided with the surface or is out of the water, $B_t = 0$ otherwise. When $B_t = -10000$ the episode is terminated.

The constants c_1, c_2, c_3, c_4 are used to scale components of the reward signal and are derived empirically.

This study will focus on the 3 MLP architectures commonly found in Reinforcement Learning literature. The 2 layer architectures studied consist of 64 nodes in each hidden layer, and 400 and 300 nodes in each hidden layer. The 3 layer architecture studied is comprised of 100, 50, and 25 nodes in each hidden layers. Each layer uses the Rectified Linear Unit (ReLU) activation function. The output of the critic network is unbounded, and the output of the actor network is bounded using a hyperbolic tangent function and then scaled appropriated for the action space. The DNN architectures are illustrated in **Figure 5** and **Figure 6**

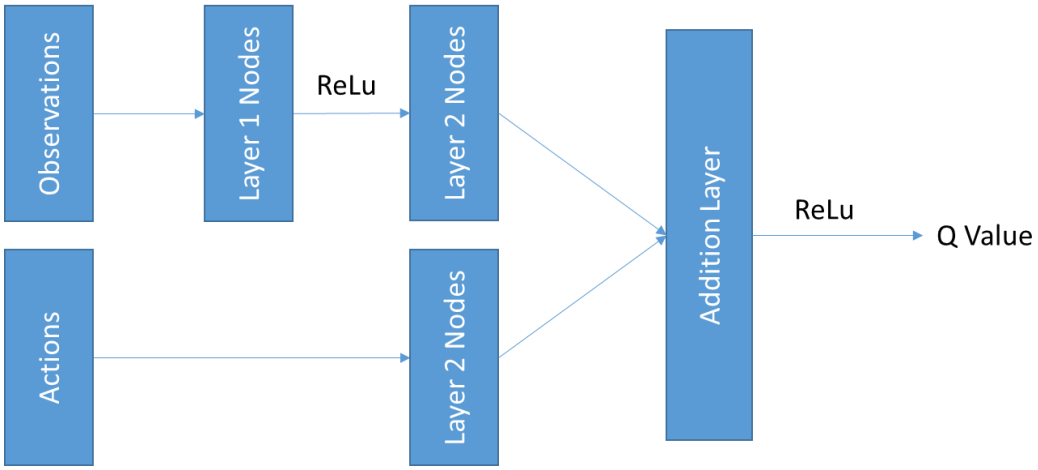

Figure 5 (a) – 2 layer critic network architecture.

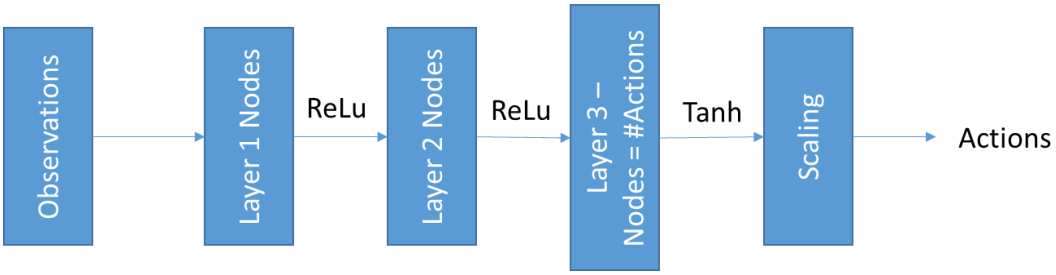

Figure 5 (b) – 2 layer actor network architecture.

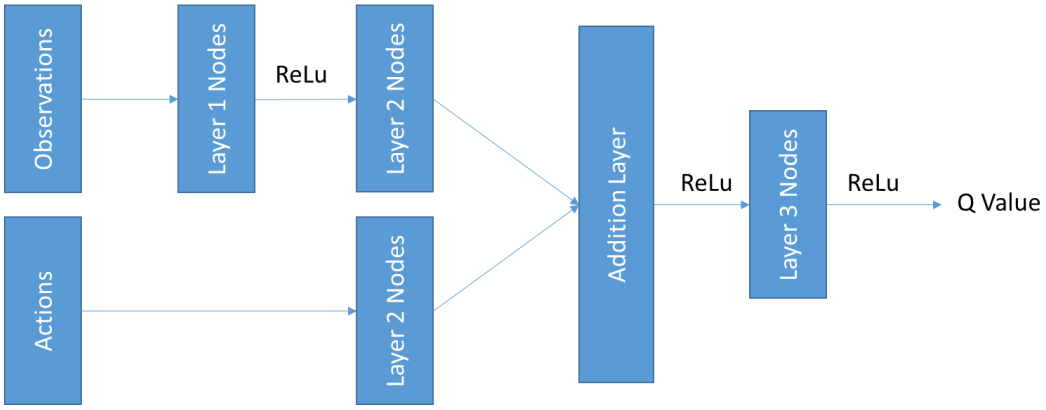

Figure 6 (a) – 3 layer critic network architecture.

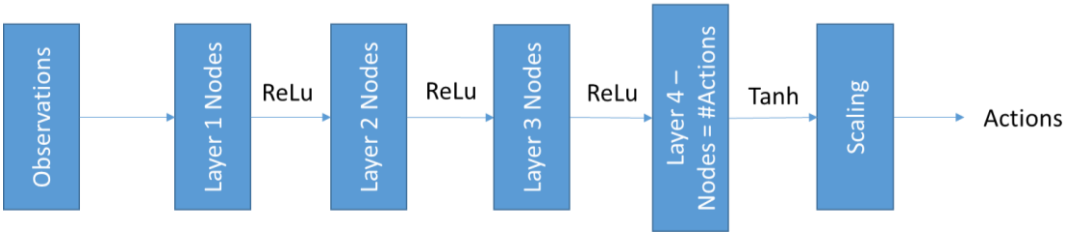

Figure 6 (b) – 3 layer actor network architecture.

The DDPG agent hyperparameters are described in **Table 1**:

Parameter Name	**Value**
Critic Learning Rate	0.001
Critic L2 Regularization Factor	0.0001
Actor Learning Rate	0.0001
Actor L2 Regularization Factor	0.0001
Target Smoothing Factor	0.001
Target Update Frequency	1
Batch Size	64
Buffer Size	1000000
Discount Factor	0.99
OU Noise Variance	$0.3\ (Action\ Space_{max} - Action\ Space_{min})$
Noise Decay Rate	0.000001
Noise Mean Attraction Constant	0.15
Noise Mean	0

Table 1 – DDPG model hyperparameters

4.2 EXPERIMENTAL RESULTS

The DDPG agent was trained until it reached an average reward value of 5,950 with a scoring window of 10 episodes. The problem is considered solved when that average reward value is achieved. Each training episode was a 600 second simulation with a timestep size of 1 second. Average reward curves for the DDPG agent for each of the DNN architectures being examined is in **Figure 7**

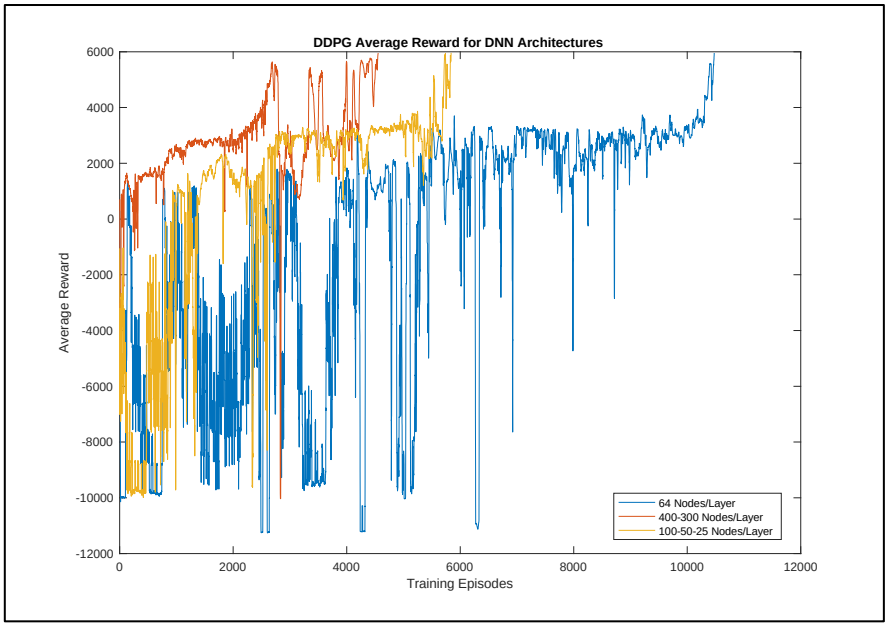

Figure 7 – Average reward for each DNN architecture during agent training

Performance of the trained DDPG agents for the each of the different DNN architectures being examined are in **Figure 8** below.

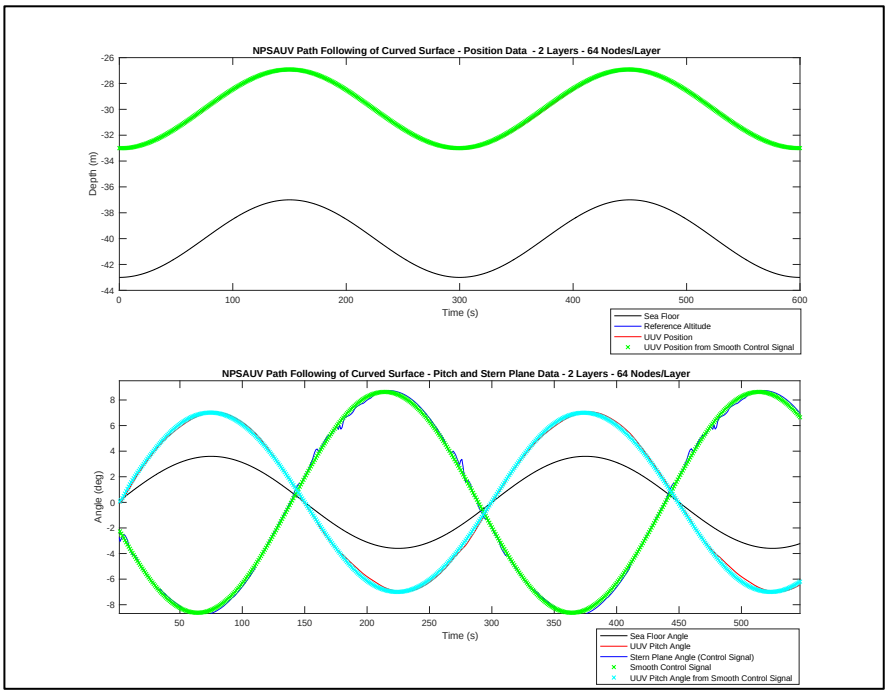

Figure 8 (a) – DDPG model performance with 2 layer DNN with 64 nodes per layer

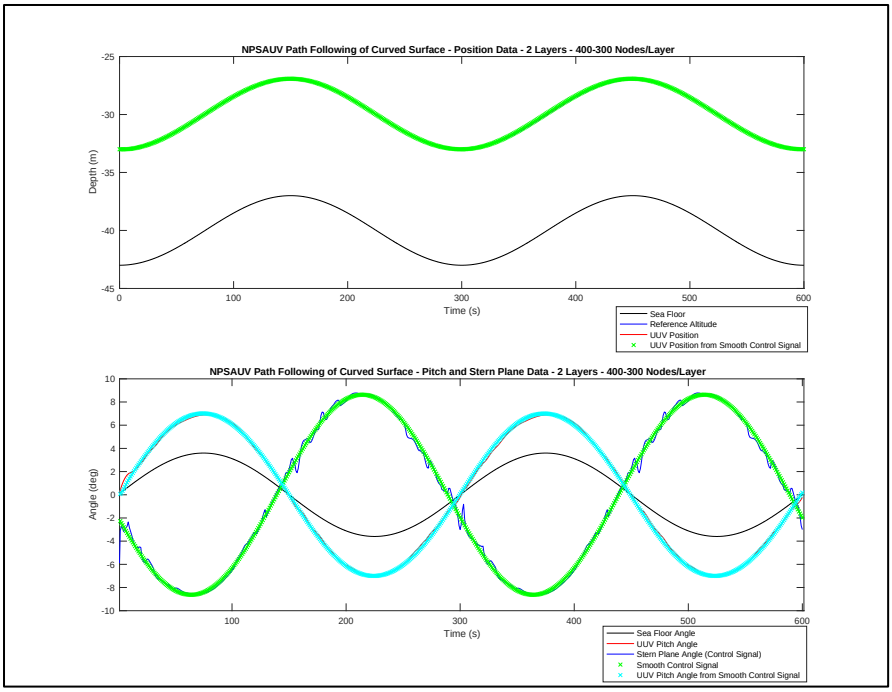

Figure 8 (b) – DDPG model performance with 2 layer DNN with 400 nodes in the first layer and 300 nodes in the second layer.

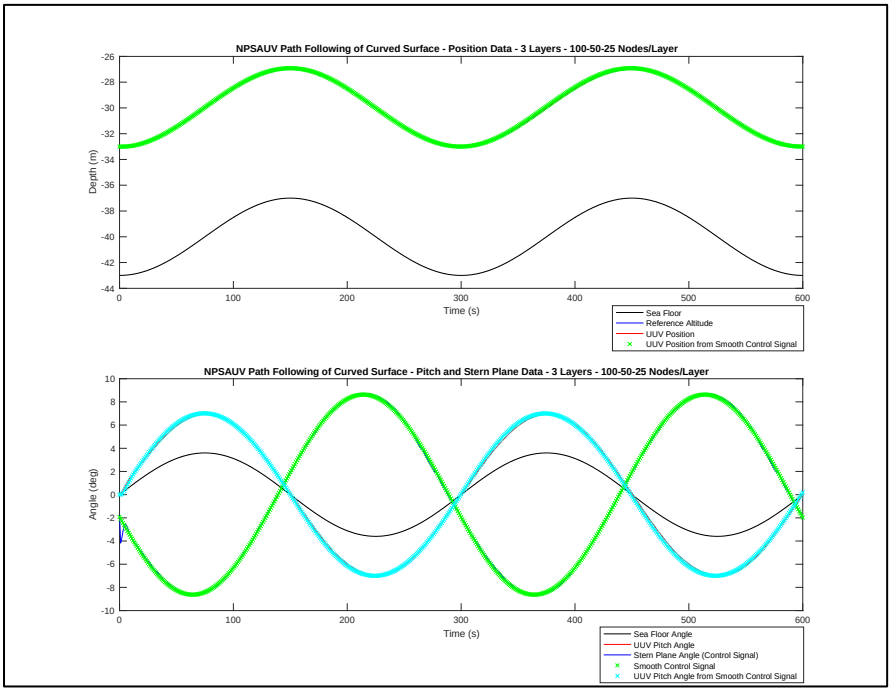

Figure 8 (c) – DDPG model performance with 3 layer DNN with 100, 50, and 25 layers in the first, second , and third layers respectively.

Upon visual inspection, the DNN with 400 and 300 nodes per layer produces a control signal with undesirable oscillations. The DNN with 64 nodes per layer produces a smoother control

signal than the DNN with 400 and 300 nodes per layer, but some oscillations exist. Finally, the DNN with 100, 50, and 25 nodes per layer produces a near perfectly smooth control signal with no oscillations.

To quantify the oscillations produced by the DNN architectures being examined, a comparison is made by computing the Mean Squared Error (MSE) between the control signal produced by the DDPG agent for each DNN being examined and a perfectly smooth control signal defined by the curve:

$$SmoothControlSignal = 8.8223e^{-4} + 8.6279sin(2\pi t + 3.3660)$$

$$MSE = \frac{1}{n}\sum_{1}^{n}(UUV\_Response_{\mathrm{DDPG},i} - UUV\_Response_{\mathrm{Smooth},i})^2$$

The UUV's pitch angle response will be examined similarly by computing the MSE between the UUV response of the DDPG agent's control signal for each DNN being examined and the smooth control signal. Additionally, the MSE of the UUV position will be examined by comparing the UUV response from the DDPG agent's control signal for each DNN architecture being examined and the reference altitude. Comparisons of the pitch angle response and UUV position will quantify the effects of the control signal oscillation. Results are in **Table 2**.

DNN Architecture	MSE - Control Signal	MSE – Pitch Angle
64-64	0.090	0.043
400-300	0.199	0.029
100-50-25	0.021	0.020

Table 2- Mean squared error comparison between control signal and pitch angle produced by the Reinforcement Learning control system and a smooth control signal.

As expected, the higher MSE values correlate to oscillations in the control signal. The values in **Table 2** support the visual inspection. Additionally, the MSE of the pitch angle is lowest for the 3 layer 100-50-25 node/layer architecture, followed by the 2 layer 64 nodes/layer and 400-300 nodes/layer architectures. All 3 architectures produced the same MSE for altitude of 0.001. This is not unexpected since altitude can be optimally controlled through bang-bang control.

The lower MSE of the control signal and pitch angle for the 3 layer 100-50-25 node/layer architecture makes this the preferred DNN architecture for this problem because it will result in more efficient use of UUV energy by eliminating unnecessary changes of the control surfaces. It will also reduce high frequency, high amplitude oscillations that can cause damage to control surface actuators and preserve sensor data quality by reducing unwanted changes to the UUV's attitude.

5. CONCLUSIONS AND FUTURE WORK

In this paper we demonstrated that there is more than just reward signal engineering and algorithmic considerations to be made when developing a control system based on Deep Reinforcement Learning methods where it is desirable to have a smooth control signal that is free of high frequency and high amplitude oscillations. We showed that the DNN architecture plays a role in the characteristics of the control signal that is produced by a DDPG agent by varying the DNN architectures while keeping the DDPG agent hyperparameters fixed. We quantified the results by comparing the UUV's response to the control signal produced by the DDPG agent to a perfectly smooth control signal and identified an optimal DNN architecture for this problem.

Although these results are presented as part of a simplified problem, the results of this study can be extended to more complex scenarios and should be considered for anyone designing control systems using Deep Reinforcement Learning methods.

In future work, we will examine additional DNN architectures and also how DNN architectures affect the control signal produced by other Deep Reinforcement Learning algorithms that work with continuous action spaces such as Proximal Policy Optimization (PPO) and Twin-Delayed Deep Deterministic Policy Gradient (TD3). We will also extend this type of study to more complex tasks and with Reinforcement Learning agents that have modified algorithms to reduce the computational time and memory requirements during training.

6. REFERENCES

1. Hsu, Y., Wu, H., You, K., & Song, S. (2019). A selected review on reinforcement learning based control for autonomous underwater vehicles. arXiv preprint arXiv:1911.11991.
2. Yu, R., Shi, Z., Huang, C., Li, T., & Ma, Q. (2017, July). Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle. In 2017 36th Chinese Control Conference (CCC) (pp. 4958-4965). IEEE.
3. Wu, H., Song, S., You, K., & Wu, C. (2018). Depth control of model-free auvs via reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 49(12), 2499-2510.
4. Bøhn, E., Coates, E. M., Moe, S., & Johansen, T. A. (2019, June). Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization. In 2019 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 523-533). IEEE.
5. Koch, W. (2019). Flight controller synthesis via deep reinforcement learning. arXiv preprint arXiv:1909.06493.
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
7. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
8. Mysore, S., Mabsout, B., Mancuso, R., & Saenko, K. (2020). Regularizing Action Policies for Smooth Control with Reinforcement Learning. arXiv preprint arXiv:2012.06644.

9. Healey, A. J., & Lienard, D. (1993). Multivariable sliding mode control for autonomous diving and steering of unmanned underwater vehicles. IEEE journal of Oceanic Engineering, 18(3), 327-339.
10. Perez, T., Smogeli, O., Fossen, T., & Sorensen, A. J. (2006). An overview of the marine systems simulator (MSS): A simulink toolbox for marine control systems. Modeling, identification and Control, 27(4), 259-275.
11. Fossen, T. I. (2011). Handbook of marine craft hydrodynamics and motion control. John Wiley & Sons.
12. Yuh, J. (2000). Design and control of autonomous underwater robots: A survey. Autonomous Robots, 8(1), 7-24. (ref 32 from NPSAUV paper)
13. Control Tutorial With Matlab and Simulink, https://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID
14. Graesser, L., & Keng, W. L. (2019). Foundations of deep reinforcement learning: theory and practice in Python. Addison-Wesley Professional.
15. Rummery, G. A., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Vol. 37, p. 20). Cambridge, UK: University of Cambridge, Department of Engineering
16. Rubí, B., Morcego, B., & Pérez, R. (2020, May). A Deep Reinforcement Learning Approach for Path Following on a Quadrotor. In 2020 European Control Conference (ECC) (pp. 1092-1098). IEEE.
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. nature, 518(7540), 529-533.
18. Uhlenbeck, G. E., & Ornstein, L. S. (1930). On the theory of the Brownian motion. Physical review, 36(5), 823.
19. Options for DDPG agent, 2020, Mathworks, https://www.mathworks.com/help/reinforcement-earning/ref/rlddpgagentoptions.html
20. Deep Deterministic Policy Gradient Agents, 2020, Mathworks, https://www.mathworks.com/help/reinforcement-learning/ug/ddpg-agents.html

7. APPENDIX

Equations of Motion for Aires UUV for each degree of freedom as described in [9] and implemented in [10]

The equation of motion in the surge direction is:

$$
\begin{aligned}
&m[\dot{u} - vr + wq - x_G(q^2 + r^2) + y_G(pq - \dot{r}) + Z_G(pr + \dot{q})] \\
&\quad = \frac{\rho}{2}L^4[X_{pp}p^2 + X_{qq}q^2 + X_{rr}r^2 + X_{pr}pr] \\
&\quad + \frac{\rho}{2}L^3\left[X_{\dot{u}}\dot{u} + X_{wq}wq + X_{vp}vp + X_{vr}vr + u\dot{q}\left(X_{q\delta_s}\delta_s + X_{\frac{q\delta b}{2}}\delta_{bp} + X_{\frac{q\delta b}{2}}\delta_{bs}\right)\right. \\
&\quad \left. + X_{r\delta r}ur\delta_r\right] + \frac{\rho}{2}L^2[X_{vv}v^2 + X_{ww}w^2 + X_{v\delta r}uv\delta_r \\
&\quad + uw\left(X_{w\delta s}\delta_s + X_{w\delta b/2}\delta_{bs} + X_{w\delta b/2}\delta_{bp}\right) \\
&\quad + u^2\left(X_{\delta s\delta s}{\delta_s}^2 + X_{\delta b\delta b/2}{\delta_{bs}}^2 + X_{\delta r\delta r}{\delta_r}^2\right) - (W - B)sin\theta \\
&\quad + \frac{\rho}{2}L^3X_{q\delta sn}uq\delta_s\epsilon(n) + \frac{\rho}{2}L^2\left(X_{w\delta sn}uw\delta_{sn} + X_{\delta s\delta sn}u^2{\delta_s}^2\right)\epsilon(n) + \frac{\rho}{2}L^2u^2X_{prop}
\end{aligned}
$$

The equation of motion in the sway direction is:

$$
\begin{aligned}
&m[\dot{v} + ur - wp + x_G(pq + \dot{r}) - y_G(p^2 + r^2) + z_G(qr - \dot{p})] = \frac{\rho}{2}L^4[Y_{\dot{p}}\dot{p} + Y_{\dot{r}}\dot{r} + Y_{pq}pq + \\
&Y_{qr}qr] + \frac{\rho}{2}L^3[Y_{\dot{v}}\dot{v} + Y_pup + Y_rur + Y_{vq}vq + Y_{wp}wp + Y_{wr}wr] + \frac{\rho}{2}L^2[Y_vuv + Y_{vw}vw + \\
&Y_{\delta r}u^2\delta_r] - \int_{x_{tail}}^{x_{nose}}[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2]\frac{v + xr}{U_{cf}(x)}xdx + (W - B)cos\theta sin\phi
\end{aligned}
$$

The equation of motion in the heave direction is:

$$
\begin{aligned}
&m[\dot{w} + uq - vp + x_G(pr - \dot{q}) + y_G(qr + \dot{p}) - z_G(p^2 + q^2)] \\
&\quad = \frac{\rho}{2}L^4[Z_{\dot{q}}\dot{q} + Z_{pp}p^2 + Z_{pr}pr + Z_{rr}r^2] + \frac{\rho}{2}L^3[Z_{\dot{w}}\dot{w} + Z_quq + Z_{vp}vp + Z_{vr}vr] \\
&\quad + \frac{\rho}{2}L^2[Z_wuw + Z_vv^2 + u^2(Z_{\delta s}\delta_s + Z_{\delta b/2}\delta_{bs} + Z_{\delta b/2}\delta_{bp})] \\
&\quad + \frac{\rho}{2}\int_{x_{tail}}^{x_{nose}}[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2]\frac{w - xq}{U_{cf}(x)}xdx \\
&\quad + (W - B)cos\theta sin\phi + \frac{\rho}{2}L^3Z_{qn}uq\epsilon(n) + \frac{\rho}{2}L^2[Z_{wn}uw + Z_{\delta sn}u^2\delta_s)\epsilon(n)
\end{aligned}
$$

The roll equation of motion is:

$$
\begin{aligned}
&I_x\dot{p} + (I_z - I_y)qr - I_{xy}(pr - \dot{q}) - I_{yz}(\dot{q}^2 - r^2) + I_{xz}(pq + \dot{r}) \\
&\quad + m[y_G(\dot{w} - uq + vp) - z_G(\dot{v} + ur - wq)] \\
&\quad = \frac{\rho}{2}L^5[K_{\dot{p}}\dot{p} + K_{\dot{r}}\dot{r} + K_{pq}pq + K_{qr}qr] \\
&\quad + \frac{\rho}{2}L^4[K_{\dot{v}}\dot{v} + K_pup + K_rur + K_{vq}vq + K_{wp}wp + K_{wr}wr] \\
&\quad + \frac{\rho}{2}L^3[K_vuv + K_{vw}vw + u^2(K_{\delta b/s}\delta_{bp} + K_{\delta b/2}\delta_{bs})] + (y_GW - y_BB)\cos\theta\cos\phi \\
&\quad - (z_GW - z_GB)cos\theta sin\phi + \frac{\rho}{2}L^4K_{pn}up\epsilon(n) + \frac{\rho}{2}L^3u^2K_{prop}
\end{aligned}
$$

The equation of motion in the pitch direction is:

$$
\begin{aligned}
I_y\dot{q} + (I_x - I_z)pr - I_{xy}(qr + p) + I_{yz}\dot{(}pq - \dot{r}) + I_{xz}(p^2 - r^2) \\
- m[x_G(\dot{w} - uq + vp) - z_G(u \dot{-} vr + wq)] \\
= \frac{\rho}{2}L^5\left[M_{\dot{q}}\dot{q} + M_{pp}p^2 + M_{pr}pr + M_{rr}r^2\right] \\
+ \frac{\rho}{2}L^4\left[M_{\dot{w}}\dot{w} + M_{uq}uq + M_{vp}vp + M_{vr}vr\right] + \frac{\rho}{2}L^3[M_{uw}uw + M_{vv}v^2 \\
+ u^2\left(M_{\delta s}\delta_s + M_{\delta b/2}\delta_{bp} + M_{j\delta b/2}\delta_{bs}\right)\Big] \\
- \int_{x_{tail}}^{x_{nose}} \left[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2\right]\frac{w + xq}{U_{cf}(x)}xdx \\
- (x_G W - x_B B)\cos\theta\cos\phi - (z_G W - z_B B)sin\theta + \frac{\rho}{2}L^4 M_{qn}uq\epsilon(n) \\
+ \frac{\rho}{2}L^3[M_{wn}uw + M_{\delta sn}u^2\delta_s)\epsilon(n)
\end{aligned}
$$

The yaw equation of motion is:

$$
\begin{aligned}
I_z\dot{r} + \left(I_y - I_x\right)pq - I_{xy}(p^2 - q^2) - I_{yz}(pr + \dot{q}) + I_{xz}(qr - \dot{p}) \\
+ m[x_G(\dot{v} - ur + wp) - y_G(\dot{u} - vr + wq)] \\
= \frac{\rho}{2}L^5\left[N_{\dot{p}}\dot{p} + N_{\dot{r}}\dot{r} + N_{pq}pq + N_{qr}qr\right] \\
+ \frac{\rho}{2}L^4\left[N_{\dot{v}}\dot{v} + N_p up + N_r ur + N_{vq}vq + N_{wp}wp + N_{wr}wr\right] \\
+ \frac{\rho}{2}L^3[N_v uv + N_{vw}vw + N_{\delta r}u^2\delta_r] \\
- \int_{x_{tail}}^{x_{nose}} \left[C_{dy}h(x)(v + xr)^2 + C_{dz}b(x)(w - xq)^2\right]\frac{v + xr}{U_{cf}(x)}xdx \\
+ (x_G W - x_B B)\cos\theta\sin\phi - (y_G W - y_B B)sin\theta + \frac{\rho}{2}L^3 u^2 N_{prop}
\end{aligned}
$$

The Euler angle rates, global positions, crossflow velocity, and propulsion terms used in the above equations of motion are:

$$\dot{\phi} = p + qsin\phi tan\theta + rcos\phi tan\theta$$

$$\dot{\theta} = qcos\phi - rsin\phi$$

$$\dot{\psi} = (qsin\phi - rcos\phi)/cos\theta$$

$$\dot{X} = u_{c0} + ucos\psi cos\theta + v[cos\psi sin\theta sin\phi - sin\psi cos\phi] + w[cos\psi sin\theta sin\phi - sin\psi sin\phi]$$

$$\dot{Y} = v_{c0} + usin\psi cos\theta + v[sin\psi sin\theta sin\phi - cos\psi cos\phi] + w[sin\psi sin\theta sin\phi - cos\psi sin\phi]$$

$$\dot{Z} = w_{c0} - usin\theta + vcos\theta sin\psi + wcos\theta sin\phi$$

$$U_{cf}(x) = [(v + xr)^2 + (w - xq)^2]^{1/2}$$

$$X_{prop} = C_{d0}(\eta|\eta| - 1);\ \eta = 0.012n/u$$

$$\epsilon(n) = -1 + sign(n)/sign(u)^*\ (\sqrt{C_t + 1} - 1)/(\sqrt{C_{t1} + 1} - 1)$$

$$C_t = 0.008L^2\eta|\eta|/2$$

$$C_{t1} = 0.008L^2/2$$

Constants uses in the equations of motion:

$W = 53.4kN$	$B = 55.4kN$	$L = 5.3m$	$I_x = 13587Nms^2$
$I_{xy} = -13.58\ Nms^2$	$I_{yx} = -13.58\ Nms^2$	$I_{xz} = -13.58\ Nms^2$	$I_y = 13587\ Nms^2$
$I_x = 2038\ Nms^2$	$x_G = 0$	$x_B = 0$	$y_G = 0$
$y_B = 0.0$	$z_G = 6.1cm$	$z_B = 0$	$g = 9.8\ m/s^2$
$\rho = 1000\ kg/m^3$	$m = 5454.54\ kg$		
$X_{pp} = 7.0e-3$	$X_{qq} = -1.5e-2$	$X_{rr} = 4.0e-3$	$X_{pr} = 7.5e-4$
$X_{\dot{u}} = -7.6e-3$	$X_{wq} = -2.0e-1$	$X_{vp} = -3.0e-3$	$X_{vr} = 2.0e-2$
$X_{q\delta s} = 2.5e-2$	$X_{q\delta b/2} = -1.3e-3$	$X_{r\delta r} = -1e-3$	$X_{vv} = 5.3e-2$
$X_{ww} = 1.7e-1$	$X_{v\delta r} = 1.7e-3$	$X_{w\delta s} = 4.6e-2$	$X_{w\delta b/2} = 0.5e-2$
$X_{\delta s\delta s} = -1e-2$	$X_{\delta b\delta b/2} = -4e-3$	$X_{w\delta s} = 4.6e-2$	$X_{q\delta sn} = 2e-3$
$X_{w\delta sn} = 3.5e-3$	$X_{\delta s\delta sn} = -1.6e-3$		
$Y_{\dot{p}} = 1.2e-4$	$Y_r = 1.2e-3$	$Y_{pq} = 4e-3$	$Y_{qr} = -6.5e-3$
$Y_{\dot{v}} = -5.5e-2$	$Y_p = 3.0e-3$	$Y_r = 3.0e-2$	$Y_{vq} = 2.4e-2$
$Y_{wp} = 2.3e-1$	$Y_{wr} = -1.9e-2$	$Y_v = -1.0e-1$	$Y_{vw} = 6.8e-2$
$Z_{\dot{q}} = -6.8e-3$	$Z_{pp} = 1.3e-4$	$Z_{pr} = 6.7e-3$	$Z_{rr} = -7.4e-3$
$Z_{\dot{w}} = -2.4e-1$	$Z_q = -1.4e-1$	$Z_{vp} = -4.8e-2$	$Z_{vr} = 4.5e-2$

$Z_w = -3.0e-1$ $Z_{vv} = -6.8e-2$ $Z_{\delta s} = -7.3e-2$ $Z_{\delta b/2} = -1.3e-2$

$Z_{qn} = -2.9e-3$ $Z_{wn} = -5.1e-3$ $Z_{\delta sn} = -1.0e-2$

$K_{wp} = -1.3e-4$ $K_{wr} = 1.4e-2$ $K_v = 3.1e-3$ $K_{vw} = -1.9e-1$

$K_{\delta b/2} = 0.0$ $K_{pn} = -5.7e-4$ $K_{prop} = 0.0$

$M_{\dot{q}} = -1.7e-2$ $M_{pp} = 5.3e-5$ $M_{pr} = 5.0e-3$ $M_{rr} = 2.9e-3$

$M_{\dot{w}} = -6.8e-2$ $M_{uq} = -6.8e-2$ $M_{vp} = 1.2e-3$ $M_{vr} = 1.7e-2$

$M_{uw} = 1.0e-1$ $M_{vv} = -2.6e-2$ $M_{\delta s} = -4.1e-2$ $M_{\delta b/2} = 3.5e-3$

$M_{qn} = -1.6e-3$ $M_{wn} = -2.9e-3$ $M_{\delta sn} = -5.2e-3$

$N_{\dot{p}} = -3.4e-5$ $N_{\dot{r}} = -3.4e-3$ $N_{pq} = -2.1e-2$ $N_{qr} = 2.7e-3$

$N_{\dot{v}} = 1.2e-3$ $N_p = -8.4e-4$ $N_r = -1.6e-2$ $N_{vq} = -1.0e-2$

$N_{wp} = -1.7e-2$ $N_{wr} = 7.4e-3$ $N_v = -7.4e-3$ $N_{vw} = -2.7e-2$

$N_{\delta r} = -1.3e-2$ $N_{prop} = 0.0$

INITIAL DISTRIBUTION LIST

Addressee	**No. of Copies**